

Title	Generating a Workflow for Change Support of UML Documents
Author(s)	Kotani, Masayuki; Ochimizu, Koichiro
Citation	Research report (School of Information Science, Japan Advanced Institute of Science and Technology), IS-RR-2003-007: 1-7
Issue Date	2003-08-26
Type	Technical Report
Text version	publisher
URL	http://hdl.handle.net/10119/8433
Rights	
Description	リサーチレポート（北陸先端科学技術大学院大学情報科学研究科）

Generating a Workflow for Change Support of UML Documents

Masayuki Kotani* and Koichiro Ochimizu*

August 26, 2003

IS-RR-2003-007

*School of Information Science

Japan Advanced Institute of Science and Technology (JAIST)

Asahidai 1-1, Tatsunokuchi

Ishikawa, 923-1292 Japan

m.kotani@jaist.ac.jp, ochimizu@jaist.ac.jp

Generating a Workflow for Change Support of UML Documents

Masayuki Kotani , Koichiro Ochimizu
Japan Advanced Institute of Science and Technology
Department of Information Science
E-mail: {m-kotani, ochimizu}@jaist.ac.jp

Abstract

In this paper, we propose a method to generate a workflow for the change of UML diagrams. We take model based translation approach to generate dependency relationships between model elements in UML diagrams. In drawing UML diagrams, a modeler or a designer should define various kinds of dependency relationships such as refine and trace, by themselves based on the traceability policy. This work is very error-prone and time consuming especially in a maintenance phase. We propose the meta model of UML to produce dependency relationships among model elements of UML diagrams automatically. We can generate a workflow for change support from the database containing UML diagrams with dependency relationships. We examined the effectiveness of our method. We found several dependency relationships not defined by the examinee.

1. Introduction

In a software development, a developer creates an artifact referring other artifacts. There are various kinds of dependencies[†] among them, such as refine and trace. In creating or changing an artifact, the developer should define those dependency relationships by themselves based on the traceability policy. This work is very error-prone and time consuming especially in a maintenance phase. In this paper, we propose a method of producing dependency relationships among model elements of UML diagrams automatically, using Meta model of UML for Change (MUC). A group of artifacts connected by dependency relationships can be considered as an order of changing artifacts. We also propose a method to generate a workflow for change support. There are three kinds of change such as “Add”, “Delete” and “Modify”. Semantics of tracing is different from each change type. We propose the method of creating the workflow for each change type, taking into account the characteristic of dependency relationships.

[†] “X depends on Y” means that we must change X if we change Y.

2. The method of generating Dependency relationships among the model elements of UML diagrams

We take model based translation approach[1] to generate dependency relationships between model elements in UML diagrams[‡] as shown in Fig. 1. An input of a translator is a group of UML diagrams and an output of the translator are dependency relationships among model elements of the UML diagrams. When the translator gets inputs, it examines correspondence between a meta element of a meta model and a model element of a UML diagram. The translator generates the dependency relationships between the model elements based on the meta relations between meta elements.

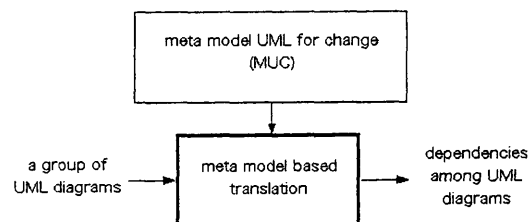


Figure 1. Outline of the method for generating dependency relationships

2.1. Meta relations to generate dependency

We show the meta relations of MUC in Table 1. In Table 1, there are three meta relations we have defined as follows;




2.1.1. Exist Together. A meta relation “Exist Together” means that when B exists together with A, we must delete B when we delete A. But, the reverse is not true. We need not to delete A when we delete B. We define its notation as an association with a solid filled diamond.

[‡] UML version 1.5[2]

2.1.2. Instance Of. A meta relation “Instance of” shows a relation between a class and an instance. We define its notation as an arrow with broken line.

2.1.3. Copy Of. “Copy Of” shows that elements of each ends are the same. We also define its notation as a broken line with two arrows.

Table 1. Kinds of meta relations

notation	meta relation
A  B	Exist Together
	Instance Of
	Copy Of

2.2. Defining meta elements

In this section, we define the meta elements of MUC.

2.2.1. Defining meta elements of UML Diagram.

We classified nine UML diagrams into three categories. There are nine UML diagrams such as a usecase diagram, a class diagram, an object diagram, a component diagram, a deployment diagram, a statechart diagram, an activity diagram, a sequence diagram and a collaboration diagram. We classified them into three based on the Classifier as shown in Table 2 where the Classifier is one of meta elements of MUC and it is an abstraction of a usecase, an actor, a class, a package, a component and a node.

Meta element “Relationship diagram” shows a relation between Classifiers. Relationship Diagram is an abstraction of a usecase diagram, a class diagram, an object diagram, a component diagram and a deployment diagram.

Meta element “Behavior Diagram” shows a behavior of the Classifier. Behavior Diagram is an abstraction of a statechart diagram and an activity diagram.

Meta element “Interaction Diagram” shows communication between Classifiers. Interaction Diagram is an abstraction of a collaboration diagram and a

sequence diagram.

Table 2. Meta elements of UML diagram

meta elements	UML diagram
Relationship Diagram	usecase diagram, class diagram, object diagram, component diagram, deployment diagram
Behavior Diagram	statechart diagram, activity diagram
Interaction Diagram	sequence diagram, collaboration diagram

2.2.2. Defining Meta Elements of UML model elements.

We define seven meta elements as an abstraction of model elements of UML diagrams. They are Classifier, Meta Object, Relation, Meta State, Meta Transition, Meta Instance and Meta Message as shown in Table 3.

Classifier, Meta object and Relation are elements of a Relation Diagram. We already explained Classifier. Meta object is a meta element and it is an abstraction of an object. Relation is a meta element and it is an abstraction of an association, a link, a dependency, a generalization and an aggregation.

Meta State and Meta Transition are elements of Behavior Diagram. Meta State is a meta element and it is an abstraction of a state and an action state. Meta Transition is a meta element and it is an abstraction of a transition, an action and an event.

Meta Instance and Meta Message are elements of Interaction Diagram. Meta Instance is a meta element and it is an abstraction of an instance. Meta Message is a meta element and it is an abstraction of message.

We define “Model Element” as a super class of all meta elements as shown in Fig. 2. The role of Model Element is to define “Copy Of” meta relation among any meta elements of MUC (Fig. 3).

2.2.3. A relation between a UML diagram and a group of meta elements.

We define a domain: “Relationship Diagram Domain” by packaging Relation Diagram, Classifier, Meta Object

Table 3. Meta elements of model elements of UML diagrams

meta elements		model elements of UML diagram
diagrams	components	
Relationship Diagram	Classifier	actor, usecase, class, package, component, node
	Meta object	object
	Relation	association, generalization, aggregation, link, dependency
Behavior Diagram	Meta State	state, action state
	Meta Transition	transition, event, action
Interaction Diagram	Meta Instance	object
	Meta Message	message

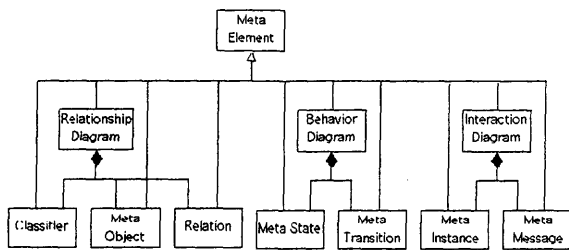


Figure 2. Meta Elements of MUC

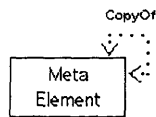


Figure 3. A recursive meta relation of Meta Element

and Relation; “Behavior diagram Domain” by packaging Behavior Diagram, Meta State and Meta Transition; “Interaction Diagram Domain” by packaging Interaction Diagram, Meta Instance and Meta Message as shown in Fig.4. The meaning of a package is to package a UML diagram and model elements in the diagram together. For an example, a statechart diagram is a UML diagram instantiated from Behavior Diagram and states and transitions in the statechart diagram are instantiated from Meta State and Meta Transition respectively. We need to package the statechart diagram, states in the diagram and transitions in the diagram together to group the model elements and the UML diagram.

2.3. Definition of the MUC

We show the MUC in Fig. 4 summarizing the consideration so far. In Fig. 4, there is a meta relation “Exist Together” between Classifier and Behavior Diagram because Behavior Diagram shows the behavior of Classifier. There is a meta relation “Instance Of” between Classifier and Meta Instance, because Meta Instance is an instance of Classifier. There is a meta relation “Copy Of” between Meta Instance and Meta Object.

2.4. Confirming the effectiveness of the MUC

We performed an experiment to confirm the effectiveness of the MUC and to improve it. We compared dependency relationships created by an examinee with ones generated by a tool. We used 15 UML diagrams described in Cruise Control & Monitoring System[3]. Results of the experiment is shown in Table 4. There are 50 dependency relationships generated by a tool. There are 42 dependency relationships that were recognized by the examinee. There are 35 dependency relationships common to both.

2.4.1. Dependencies that cannot be generated automatically but recognized by the examinee. There are 7 dependency relationships that cannot be generated by the tool. But all of them have the same type. They are the dependency relationships between a usecase diagram and a collaboration diagram. The examinee created them as the dependency relationships between a usecase and a collaboration diagram. In MUC, we need the meta relation between Classifier and Interaction Diagram to generate those dependency relationships automatically (Fig. 5).

This problem is solved if we add a meta relation between Classifier and Interaction Diagram in Fig. 5. We can instantiate a dependency relationship between a usecase and a collaboration diagram based on the meta relation. But it causes another problem. There is another meta relation between Classifier and Meta Instance in Fig. 5. It can instantiate a dependency relationship between a class and an instance in base level. Existence of two meta relations enable the tool create wrong dependency between a class and a collaboration diagram. We divide Classifier into two, Static Classifier and Dynamic Classifier (Fig. 6). Static Classifier has a meta relation “Instance Of” with Meta Instance and Dynamic Classifier has a meta relation “Exist Together” with Interaction Diagram. We need a super class Classifier of both to keep the meta relation between Classifier and Behavior Diagram. We show the improved meta model in Fig. 6.

Table 4. The result of the examination

	Automatically	Manually
A number of dependencies	50	42
A number of matched dependencies	35	

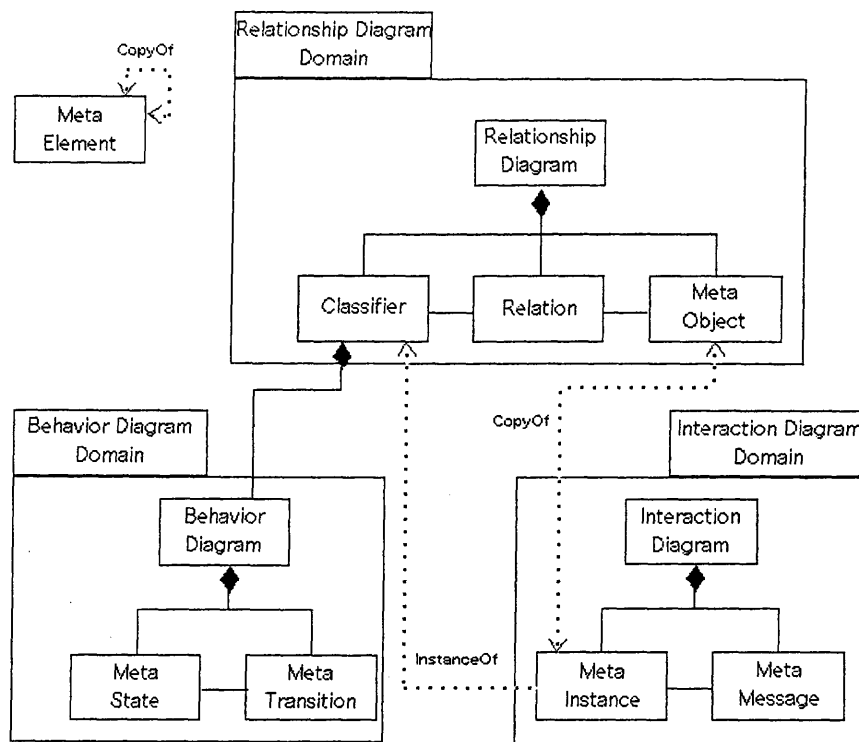


Figure 4. Meta model of UML for Change

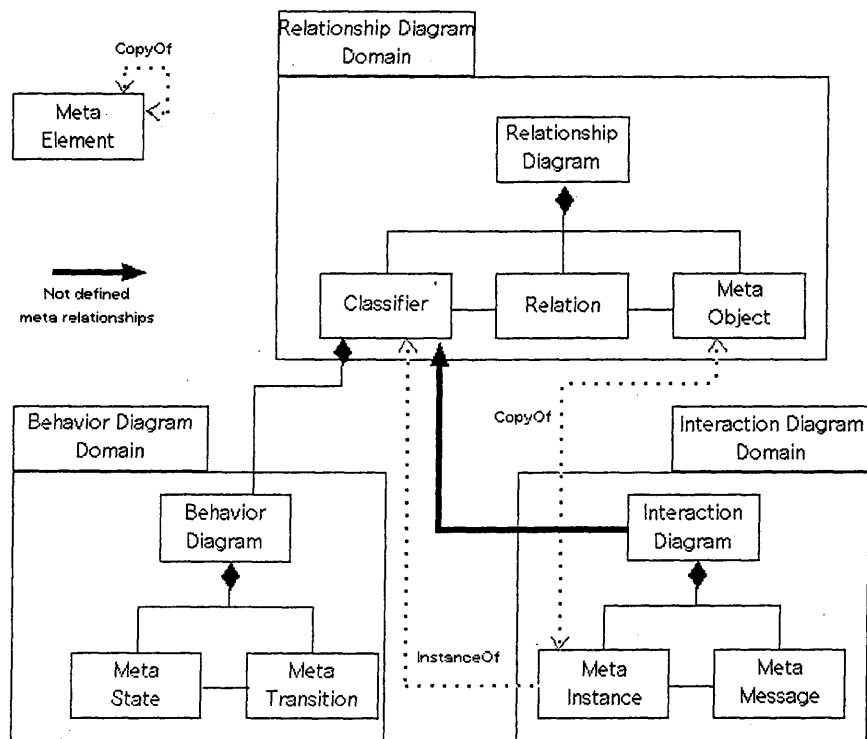


Figure 5. Meta relation between Classifier and Interaction Diagram

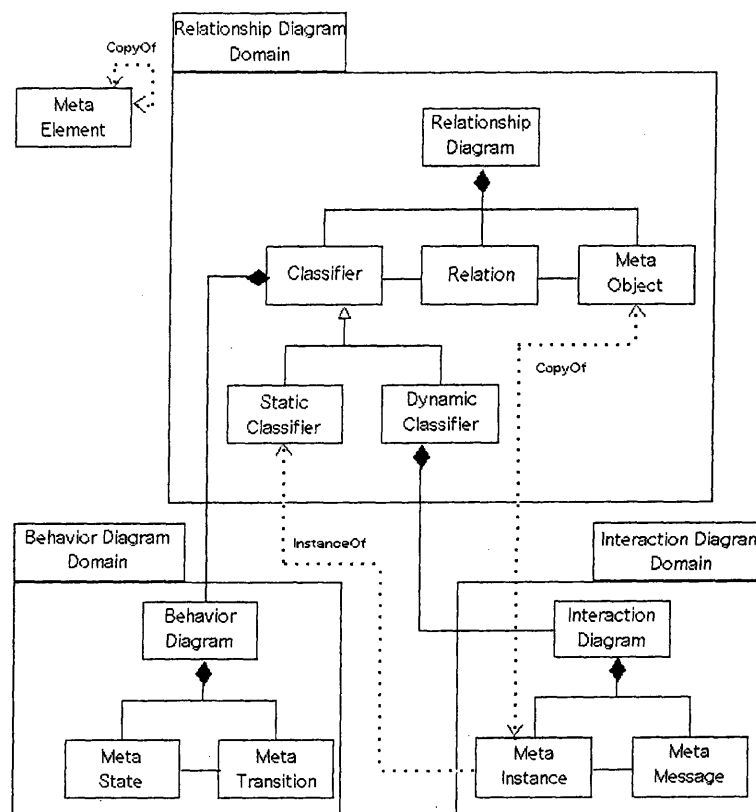


Figure 6. Improved MUC

2.4.2. The dependency relationships generated by a tool, but not created by the examinee. The examinee failed to define a lot of dependency relationships between the same model elements. As shown in Fig. 7, some class in a class diagram is copied to another class diagram. All of them should have dependency relationships with the same collaboration diagram, depending on his policy or his careless miss. The tool, however can create all of the necessary dependency relationships completely. This is one of advantages of our method.

3. Generating a workflow automatically

In this section, we will discuss how we can generate a workflow for change support. We can show an outline of our system which can generate the workflow as shown in Fig. 8. Inputs of a generator are: dependency relationships; the original UML diagrams and their model elements; change part of the UML diagram. The output of the generator is a workflow.

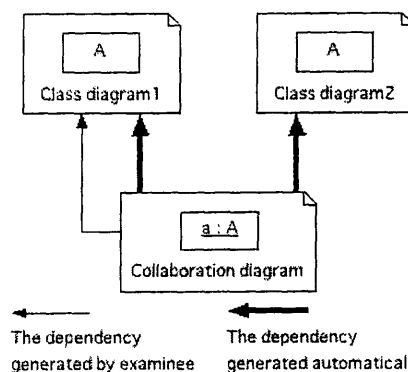


Figure 7. Dependency relationships that are generated automatically, but not created by the examinee

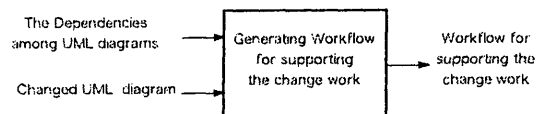


Figure 8. An Outline of the method for generating a workflow

3.1. Reexamination of semantics of meta relations from a viewpoint of change

We reexamine semantics of meta relations from the viewpoint of change support.

A meta relation "Exists Together" shows that an obeying one is deleted by deletion of an obeyed one[§].

A meta relation "Instance Of" shows that an instance is deleted when a class is deleted.

A meta relation "Copy Of" shows that a meta element of one end is the same with the other end. When one meta element is modified, the other should be modified.

Meta relation can have the following four characteristics.

- **obey**: one is obeying element, and the other is obeyed one
- **delete**: If one is deleted, the other should be deleted
- **attribute name**: both sides have the same attributes
- **attribute value**: both sides have the same attributes and their values

We show that which characteristics each meta relation has in Tab 5. For an example, "Exist Together" has two characteristics, "obey" and "delete".

3.2. Generating dependency relationships for each change type

We define usage of the characteristics to generate dependencies for tracing to each change type such as Delete and Modify

3.2.1. Delete. We generate dependency relationships only for the meta relations, "Exist together" and "Instance Of" because change type Delete has two characteristics, "obey" and "delete".

3.2.2. Modify. We generate dependency relationships only for the meta relations, "Instance Of" and "Copy Of" because change type Modify has at least one characteristic among "attribute name" and "attribute value".

Definition for change type Add is under consideration. We show relations between change types and meta

relations in Table 6.

Table 6. Related meta relations for each change type

change type	related meta relations
Delete	Exist Together, Instance Of
Modify	Instance Of, Copy Of

3.3. Generating a workflow for change

In this section, we show simple examples of workflow generation. We use four UML diagrams as an example of a workflow generation. They are: two class diagrams; a collaboration diagram; and a statechart diagram. A translator finds meta relations among model elements as shown in Fig 9.

Suppose that "Cruise Control" in "speed control" diagram is deleted. A workflow generator generates dependency relationships based on meta relations "Exist together" and "Instance Of" as shown in Fig 10.

We show another example in Fig. 11. Suppose "Cruise Control" in "speed control" diagram is modified, the workflow generator generates dependency relationships based on meta relations "Instance Of" and "Copy Of" as shown in Fig. 11.

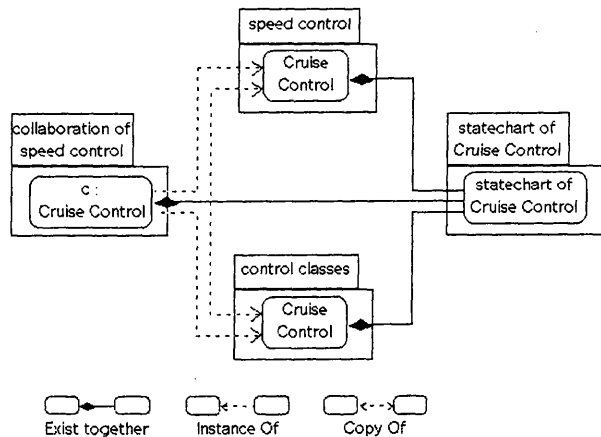


Figure 9. an example of model elements and meta relations among them

Table 5. Characteristics of meta relations

	obey	delete	attribute name	attribute value
Exist Together	○	○	—	—
Instance Of	○	○	○	—
Copy Of	○	—	○	○

○ : has a characteristic, — : not has a characteristic

[§] obeyed one obeying one

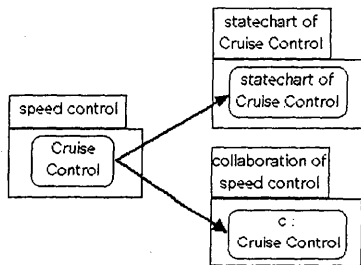


Figure 10. Generated workflow for deleting "Cruise Control" in speed control diagram

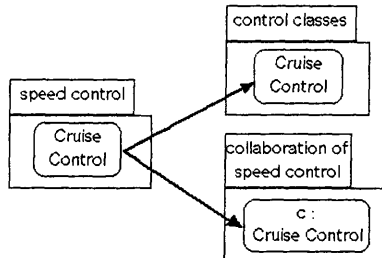


Figure 11. Generated workflow for modifying "Cruise Control" in speed control diagram

4. Supporting Tool

In this section, we explain a prototype system of our tool. The tool has three functions: creating meta elements from UML Diagrams; generating meta relations; and generating a workflow for change. This tool shows a list of model elements (Fig. 12) and the workflow for changing support (Fig. 13).

The tool generates dependency relationships by using the information created by a CASE tool. They are project files, information about diagrams, and XMI. We used IIOSS[4] as the CASE tool that is functionally compatible for Rational Rose. The tool shows a list of model elements as shown in Fig. 12 and a user can select one from the list. Then the tool generates a workflow for change as shown in Figure 13.

5. Conclusion and Future Works

In this paper, we proposed a method of generating dependency relationships among model elements of UML diagrams, and generating a workflow for change support.

We must refine our meta model especially for meta relations by abstracting the all of dependencies in UML such as derivation, refinement, trace, binding, extend, become, copy, include, instanceof, powertype, access, friend, import, call, instantiate, parameter and send[5].

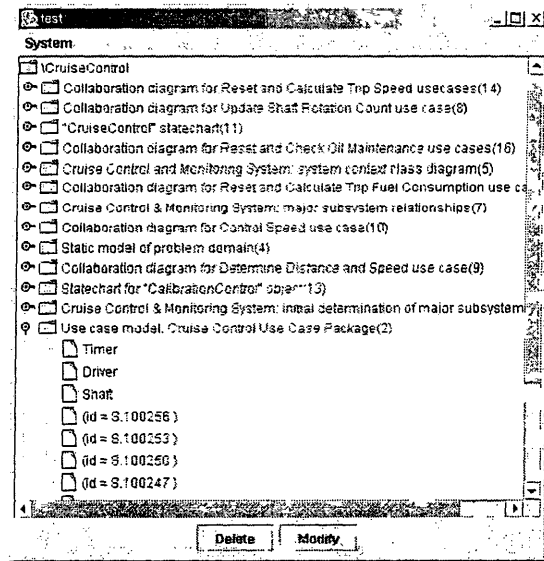


Figure 12. List of model elements

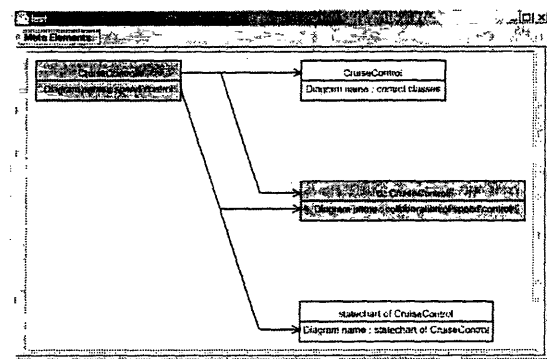


Figure 13. an example of a generated workflow

We are also developing a theory and a tool to generate a workflow that can handle multi-threaded changes.

Reference

- [1] Dragan Milićev: "Automatic Model Transformations Using Extended UML Object Diagrams in Modeling Environments", IEEE Trans. Software Eng., vol. 28, no. 4, pp.413-431, April. 2002.
- [2] Object Management Group: "Unified Modeling Language (UML), version 1.5", <http://www.omg.org/technology/documents/formal/uml.htm>
- [3] Hassan Gomaa: "Designing Concurrent, Distributed, and RealTime Applications with UML", Addison-Wesley, 2000.
- [4] The IIOSS Consortium.: "IIOSS", <http://www.iioss.org/>
- [5] Grady Booch: "The Unified Modeling Language User Guide", Addison Wesley Longman, Inc, 1999.